



## Laboratory for Data Science

*Ondrej Krasnansky, 665131*  
*Sandro Cantisano Martino, 660874*

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Data warehouse building</b>	<b>2</b>
2.1	Data Understanding and main idea . . . . .	2
2.2	Gun and participant table . . . . .	2
2.3	Geography and dates . . . . .	3
2.4	Custody . . . . .	4
2.5	Populating . . . . .	5
<b>3</b>	<b>SSIS</b>	<b>7</b>
3.0.1	Assignment 0 . . . . .	7
3.0.2	Assignment 1 . . . . .	8
3.0.3	Assignment 2 . . . . .	9
<b>4</b>	<b>OLAP Cube and MDX</b>	<b>10</b>
4.0.1	Assignment 1 . . . . .	10
4.0.2	Assignment 2 . . . . .	12
4.0.3	Assignment 3 . . . . .	13
4.0.4	Assignment 4 . . . . .	14
4.0.5	Assignment 5 . . . . .	15

# 1 Introduction

The first part of the work for the Laboratory for Data Science project is related to the building of a data warehouse starting from the file "*Police.csv*", which contains the main body of data afferent to gun violence incidents between January 2013 and March 2018 in the USA. This file contains 170929 records of crimes with following 11 columns which will be analysed, understood and elaborated in order to build a new data warehouse using python and Microsoft SQL Management Studio. The assignments contains also other two files which are *dates.xml* and 3 .json files which will be used in the last subsection for creating a measure in our fact table. The .xml file contains the information about the incident dates and how we have converted this file will be explained in the following parts.

Column Name	Description
custody_id	Identifier for custody records
gun_stolen	Indicates whether a gun was stolen
participant_age_group	Child 0-12, Teen 13-17, Adult 18+
participant_gender	Male , Female
participant_status	Killed, Unharmed, Injured, Arrested
participant_type	Type of participant (e.g., suspect, victim)
gun_type	Type of gun involved in the incident
incident_id	Identifier for the incident
date_fk	Foreign key referencing the date of the incident
latitude	Latitude coordinate of the incident location
longitude	Longitude coordinate of the incident location

Table 1: Column Names in Police.csv

## 2 Data warehouse building

### 2.1 Data Understanding and main idea

In order to build a new data warehouse we had to focus on every column and feature and understand its meanings such that would be possible to create a meaningful and functional separate tables ready to be uploaded to our database. The main approach used for this task was to check the dataset using python where the *csv* library was the main instrument to do so.

During the data understanding part it was checked a potential missing values in the dataset which was not detected for any column. In that way was not necessary to check missing values for every separate table (with exception of geography which will be explained in the next section).

In the following sections, we will explain how we have built tables: dates, geography, participant, gun and custody which will be the fact table of the data warehouse.

### 2.2 Gun and participant table

This tables can be considered the most easiest one to analyze and separate from Police.csv. The task was to separate unique tuples for each row and store them to a new csv file which will be uploaded to a data warehouse. It was developed a python function *create\_table* shown in the figure 1 which permitted us to separate the needed columns and assigning an id for a every row. This *id* will be a **primary key** for the certain table which means it identifies the unique tuple. To be able to iterate and write inside

of the file we have used *DictReader* and *DictWriter* from *csv* library. This tool permitted us to easily access the column and work with dictionaries in easy and understandable way. With this approach we have created two different csv files *gun.csv* with 80 records and 3 columns **gun\_id**, **gun\_stolen**, **gun\_type** and *participant.csv* with 49 records and 5 columns *participant\_it*, *participant\_age\_group*, *participant\_gender*, *participant\_type*, *participant\_status*.

```
def create_table(input_file, columns, id_name, output_file):
    with open(input_file, 'r', newline='') as csvfile:
        reader = csv.DictReader(csvfile)
    with open(output_file, 'w', newline='') as csv_output:
        writer = csv.DictWriter(csv_output, fieldnames=[id_name] + columns)
        writer.writeheader() # adding the column names (header)
        row_set = set()
        id_count = 0 # counter to create an id
        id_dict = {} # dictionary to assign an id number for every row
        row_dict = {}
        for row in reader:
            row_tuple = tuple(row[column] for column in columns) # every row will be tuple of each column (column1_value, column2_value)
            if row_tuple not in row_set: # if this tuple is in the set (no duplicates) id will be assigned and new dictionary created
                id_count += 1
                id_dict = {id_name: id_count}
                row_set.add(row_tuple)
                row_dict = dict(zip(columns, row_tuple)) # k: v for k, v in zip(keys, values)
                row_dict.update(id_dict)
            writer.writerow(row_dict) # create a new csv file
```

Figure 1: Create\_table function

```
from functions import create_table
#create_table(input_file, columns, id_name, output_file)
create_table('Police.csv', ['participant_age_group', 'participant_gender', 'participant_status', 'participant_type'], 'participant_id', 'participant.csv')
```

Figure 2: Create\_table example

## 2.3 Geography and dates

To extract the table for geography we had to firstly use `create_table()` function for the *latitude* and *longitude* and in that way we have created the file with unique tuples of coordinates. In that way was possible to extract cities, states and county for every coordinate. To extract this information we have used 2 different free APIs (OpenStreetNominatim and BigDataCloud) and *reverse\_geocoder* library. The approach used was to initiate the extraction using *OpenStreetNominatim*, as this library did not contain all the information especially about the cities if the city state or county was not found the code tried to extract it from *BigDataCloud*. As the information was still not complete we have tried to use *reverse\_geocoder* library to refill and complete the information. We are aware that this approach was not the best in terms of time efficiency as the whole process took around 16 hours. After the first implementation, better solution was found using just *reverse\_geocoder* and passing a list of all its coordinates (not to geocode row by row) and output it to a csv. However, after testing both solutions we have found that the first approach is more accurate in terms of location, therefore it was decided to keep it and go forward with this solution. In the python code we provide both possible solution but we go forward with *geography\_with\_id* generated by the first solution using both API and *reverse\_geocoder*.

As for the previous tables, we have generated *geo.id* as its *primary key*. For the whole process was used *DictReader* and *DictWriter* from *csv* library. Another important fact to mention, is that for column *country* was found 175 missing values. We have decided to keep these values blank as after an analysis of these records we have found that only country missing was for Washington DC which has not its own county and the second possible missing value was for City of New York. As City of New York has 5 different countries we could not fill these values with a proper and true information so it was decided to keep it without them.

Dates table was created parsing the *dates.xml* file using *xml.etree.ElementTree* library which has permitted to us extract dates in form *Year-Month-Day Hour-Minute-Second* as a string and *date\_pk* which is the primary key of the dates table and its already contained in *Police.csv*. It was used the "for" cycle to store dates and *date\_pk* to a lists in order to manage them more easily, this function was called *parse\_dates.xml()*. In the second function *get\_date\_csv()* we have implemented the writing a new csv with 7 columns (*date\_id, date, year, moth, day, day\_of\_the\_week* where all the records, with exception of *date\_id*, were stored as strings . It was decided to drop the part ('Hour-minute-second') as it was observed the same value 00:00:00 in every record. To create a new column *quarter* was used *datetime* library which permits an easy manipulation and access to this format of date. For this code was used *csv.reader* and *csv.writer* which in difference of *DictReader* returns rows as a lists. We have considered more comfortable to use this manipulation in this case.

## 2.4 Custody

To generate a final custody table, which is the fact table with which all the tables are connected. Since we are building a **star schema**, we have no connections between dimensions but all dimensions (tables previously created) are connected to the main fact table which contains all primary keys of them. In the fact table, these keys are foreign keys. As the file *Police.csv* already contains *custody\_id* we will not generate any id for this table. Another significant and important observation is that we have not created any table **incident\_id** as it would be useless, it is therefore included in the fact table **custody** where it is considered as *Degenerated Dimension (DD)*.

The fact table also includes a measure called *crime\_gravity*, which was calculated by this equation:

$$\text{crime gravity}(x) = F(x.\text{participant\_age}) \cdot F2(x.\text{participant\_type}) \cdot F3(x.\text{participant\_status})$$

Using *DictReader*, we could easily compute this multiplication for every row and create a new column with its values.

To create *custody.csv* we have opened all the files (except dates which was already contained) generated in the previous sections. Creating new 3 dictionaries (gun, participant, geography) we were able to track tuples and its corresponding id. In that way we could use *DictWriter* and iterating in *Police.csv* we have written the corresponding id for every tuple.

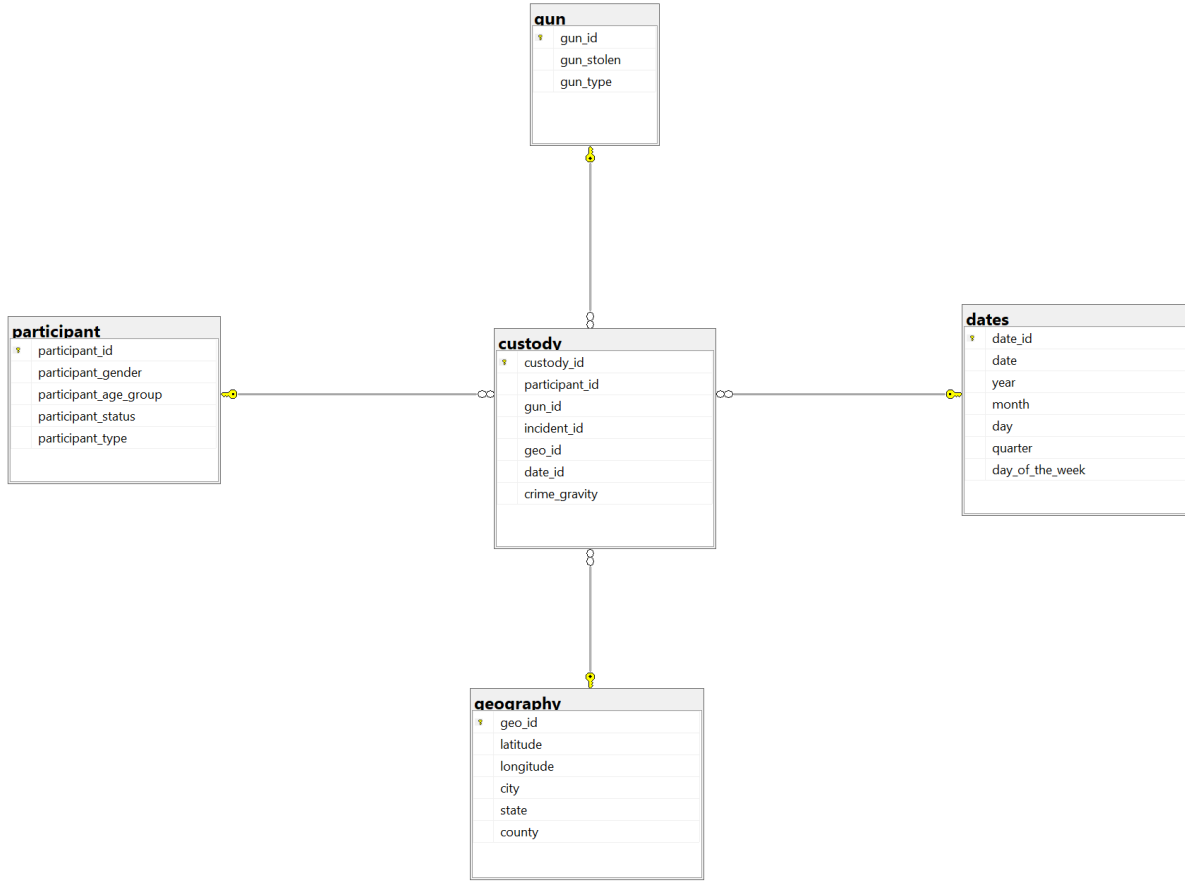


Figure 3: DWH scheme

## 2.5 Populating

The final part of this section was to populate the database on our server using python and pyodbc library. Firstly, we have created following tables manually in SQL Management Studio: *custody*, *dates*, *gun*, *participant*, *geography*. To create these tables we had to decide which data types to choose for every column in every table. As the fact table contains just id's the data type chosen was int (integer) for every column.

Column Name	Data Type
date	date
year	varchar(4)
month	varchar(2)
day	varchar(2)
quarter	varchar(2)
day of the week	varchar(20)

Table 2: Data Types for Date Columns

The main idea was to try to reduce space and put a minimum of characters possible to a varchar. As we know that, for example month takes a maximum of 2 characters (from '00' to '31') was used just varchar(2).

For the table *geography* was chosen to use varchar(100) as 60 was tried and it was not enough in some records, we have chosen to put this number a little higher as it can vary much more than in the table date. For *tables gun and participant* was chosen to use varchar(20) as this value should not vary as for geography. The final part was populating the database which was done by creating a function **populate\_table** shown in the figure 4

```
import pyodbc
import csv

#connect to data source, using userName and userPWD
import csv
import pyodbc
server = 'tcp:131.114.72.230'
database = 'Group_ID_137_DB'
username = 'Group_ID_137'
password = '8ZZ6RXQR'
connectionString = 'DRIVER={ODBC Driver 17 for SQL Server};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password
cnxn = pyodbc.connect(connectionString)
cursor = cnxn.cursor()

# Function to populate tables the database
# then just call for example populate_table('gun.csv', sql_query = INSERT...)

def populate_table(input_file, sql_query):
    cnxn = pyodbc.connect(connectionString)
    cursor = cnxn.cursor()
    with open(input_file, 'r', newline = '') as csvfile:
        reader = csv.reader(csvfile)
        next(reader)
        for row in reader:
            cursor.execute(sql_query, *row)
        cnxn.commit()
    cursor.close()
```

Figure 4: Populate\_table function

```
from functions import populate_table
# populate dates
populate_table(input_file = 'dates.csv', sql_query = "INSERT INTO dates(date_id,date,year,month,day,day_of_the_week, quarter) VALUES(?,?,?,?,?,?,?)")
```

Figure 5: Populating example

As a final result of all sections above it is presented the star scheme of the created data warehouse. The last step was to manually create relations with the corresponding *Custody fact table* and as can be seen in the figure 3, all tables are connected to the unique fact table with all surrogate keys.

## 3 SSIS

In this second part, we show the execution methods that we ran in order to solve some problems on the database previously created. To carry out these tasks we use the SSIS package of Visual Studio.

### 3.0.1 Assignment 0

The request for this first assignment was as follows:

*For every year, compute the number of total custodies.*

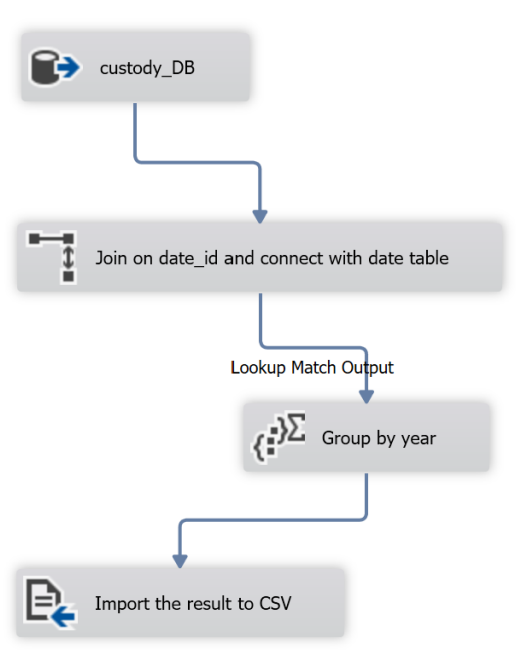


Figure 6: Assignment 0

As showed in figure 6, we first connected to the remote database by accessing the fact table "custody" and, through the **node lookup**, we mapped the records contained in Custody with the tuples contained in the table "Dates". The intent is to search for couples who are equal with the intention of representing incidents that occurred in a specific year. The **Aggregate** node allowed us to do a "group by" that for every year was counted the number of custodies, using **COUNT \*** as we are interesting in every possible custody\_id not just distinct ones. After that, the result was passed to the **Flat File Destination node** for viewing.



### 3.0.2 Assignment 1

The request for this second assignment was as follows:

*A state is considered to have a youth criminal problem if the age group with the highest overall gravity consists of individuals under 18. List every state with a youth criminal problem.*

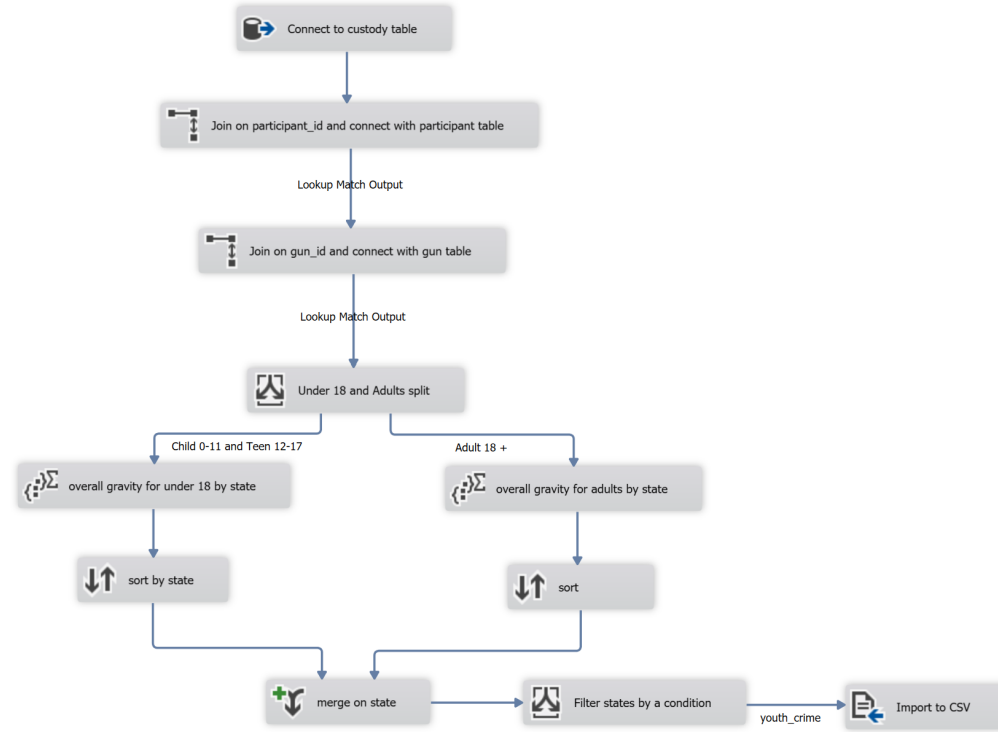


Figure 7: Assignment 1

This assignment was a little more complicated as the previous one as we had to join 2 tables to custody in order to be able to manipulate guns and participants for that it was used Lookup node as in the previous assignment. We have decided to split the data, using **conditiona split**, to cases Under 18 and Adult 18+ which means just Child 0-12 and Teen 13-17 and the rest are just custodies committed by adults. In that way we could "Group By" state and do a sum of their crime gravities. In that was easy to manipulate the data and do the comparison and report just states where the crime gravity was higher for Under 18. Firstly we had to use node of *Merge Join* which requieres sorted data which was sorted on states as the join was made also on states. In this way we have obtained states with their crime gravity for adults and under 18. Finally, was used another split node just to separate on condition to report just states where crime gravity is higher for under 18. Fortunately, this file resulted to be empty.

### 3.0.3 Assignment 2

*For each incident, compute the ratio between the total gravity of crimes with a stolen gun and the total gravity of crimes with a not-stolen gun.*

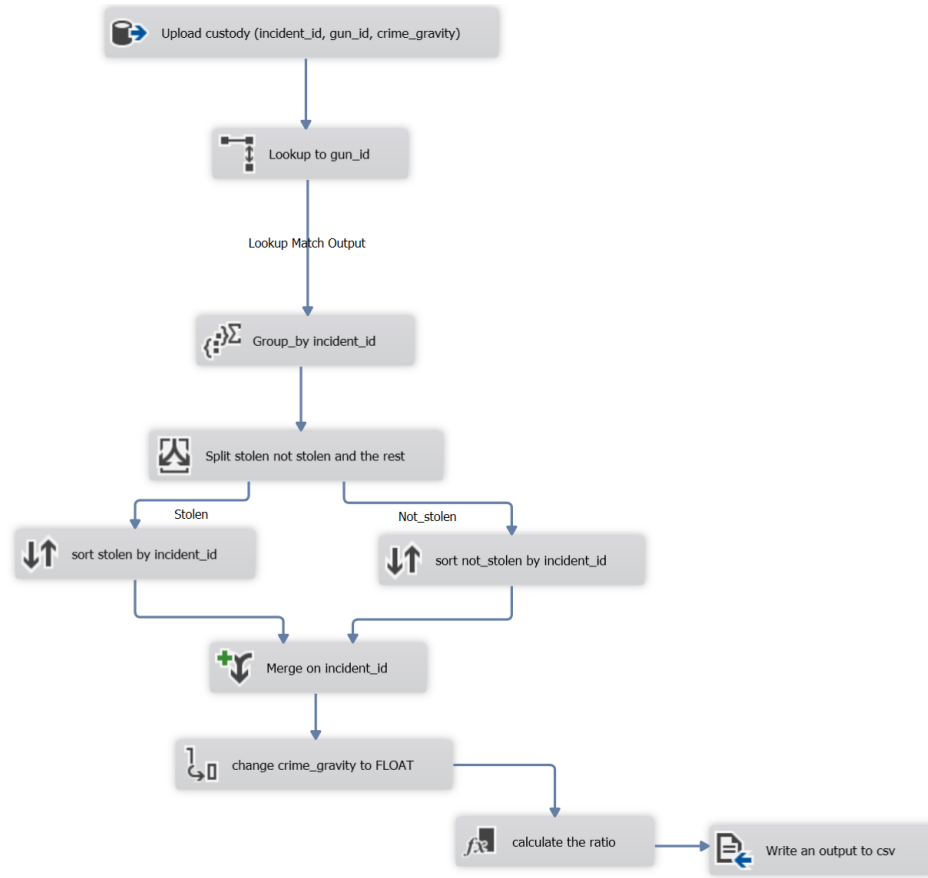


Figure 8: Assignment 2

For this assignment we have proceeded in the similar way as in the previous one with some changes. As for all of assignments we have used an OLE DB node to connect with custody fact table then using lookup we have done a join to gun table. As there were no need to group by twice we have done just one node of aggregation to group by by *incident\_id* afterwards, using **Conditional Split** the data was splitted to two flows: *stolen* and *not\_stolen* and *unknown* and *irrelevant* was not considered. This choice permitted us to avoid dividing by zero. Having the data organized by incident\_id and splitted, as in the previous task it was necessary to use **Merge Join** node to merge two different flows and always sorting by incident\_id as merge operation it requires. **Data Conversion** node was used just to convert crime\_gravity to float data type as the last step is calculating the ratio between two integer numbers. This step was done by **Derived Column**. As for all of assignments the result was ouputed to CSV file.

## 4 OLAP Cube and MDX

In this section, we discuss the creation of the Olap Cube. After creating the database, as explained in section 2, we were able to create the OLAP cube creating 5 dimensions : **Dates**, **Gun**, **Participant**, **Geography** and **Incident**. For dimensions Dates and Geography were created following hierarchies :

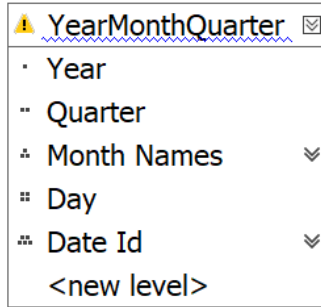


Figure 9: Dates Hierarchy

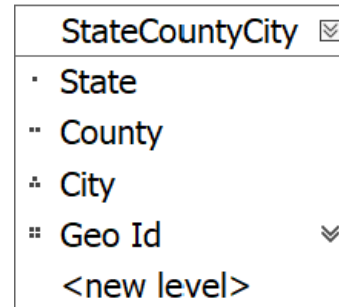


Figure 10: Geography Hierarchy

Important thing to mention is the decision to create a new column **month\_names** with the functional dependency to **month** which permits us to have a better visualization as in this case we are not using just numbers but names of months. The creation was used with following query displayed in the figure 11:

```
case Month
when 01 then 'January'
when 02 then 'February'
when 03 then 'March'
when 04 then 'April'
when 05 then 'May'
when 06 then 'June'
when 07 then 'July'
when 08 then 'August'
when 09 then 'September'
when 10 then 'October'
when 11 then 'November'
else 'December' end
```

Figure 11: month\_names creation

To conclude the cube we had to create a new dimension **Incident** which was included in the fact table as a degenerate dimension and for that reason it was necessary to specifically create it separately as the dimension custody was not created.

### 4.0.1 Assignment 1

This assignment requires to show *the percentage of growth and decrease of total crime gravity compared to the previous year for each state.*

This MDX query is designed to analyze the severity of crimes by state and year. As you can see from figure 12, for the needs to identify the year before the observed one, let's create a **PrevCrimeGravity** member to help our purpose, through the "lag(1)" function. A second **PercentageChange** member is created to calculate the percentage of change between the current Crime Gravity and that of the previous year. The member just quoted will iteratively subtract the Crime gravity and the member "PrevCrimeGravity", all divided by the latter one. The condition "IFF" handles cases where the denominator is zero, avoiding errors.

```

with member PrevCrimeGravity as ([Dates].[Year].CurrentMember.lag(1), [Measures].[Crime Gravity])

member PercentageChange as IIF([Measures].[PrevCrimeGravity] = 0,NULL,
    ([Measures].[Crime Gravity] - PrevCrimeGravity) / PrevCrimeGravity),format_string = "Percent"

select {[Measures].[Crime Gravity], PrevCrimeGravity, PercentageChange} on columns,

Generate([Geography].[State].[State],[[Geography].[State].CurrentMember} * [Dates].[Year].[Year] ) on rows
from [Police_Group_137]

```

Figure 12: Query MDX, first

So, we decide to display on the columns the members just described and the crime gravity, while on the rows we create a list of all the possible combinations(using the function **"Generate"** in the line 8) between states and years on the rows, allowing us to perform a detailed and ordered analysis by year increment state, iteratively.

Messages		Results		
		Crime Gravity	PrevCrimeGravity	PercentageChange
Alabama	2013	(null)	(null)	(null)
Alabama	2014	363	(null)	(null)
Alabama	2015	194	363	-46.56%
Alabama	2016	925	194	376.80%
Alabama	2017	2030	925	119.46%
Alabama	2018	471	2030	-76.80%
Alaska	2013	(null)	(null)	(null)
Alaska	2014	45	(null)	(null)
Alaska	2015	331	45	635.56%
Alaska	2016	503	331	51.96%
Alaska	2017	478	503	-4.97%
Alaska	2018	60	478	-87.45%
Arizona	2013	4	(null)	(null)
Arizona	2014	199	4	4875.00%
Arizona	2015	259	199	30.15%
Arizona	2016	625	259	141.31%
Arizona	2017	967	625	54.72%
Arizona	2018	256	967	-73.53%
Arkansas	2013	(null)	(null)	(null)
Arkansas	2014	209	(null)	(null)
Arkansas	2015	219	209	4.78%

Figure 13: First query MDX, output

## 4.0.2 Assignment 2

This assignment requires you to show *For each gun, show the total crime gravity in percentage with respect to the total crime gravity of all the guns.*

```
with member Gravity as IIF([Measures].[Crime Gravity] is null, 0, [Measures].[Crime Gravity])

member TotalCrimeGravity as ([Gun].[Gun Type].[All], [Measures].[Crime Gravity])

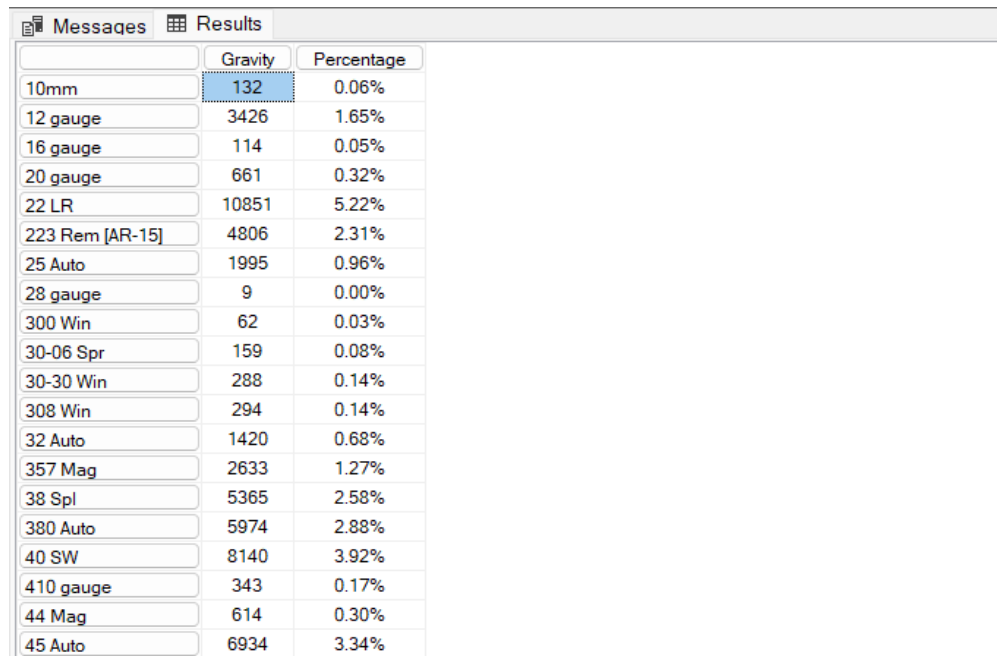
member Percentage as IIF(TotalCrimeGravity = 0, NULL,
Gravity / TotalCrimeGravity), format_string = "Percent"

select nonempty([Gun].[Gun Type].[Gun Type]) on rows,
{Gravity, Percentage} on columns

from [Police_Group_137]
```

Figure 14: Query MDX, second

This MDX query is designed to analyze data on "Crime Gravity" and firearms by type of firearm. For this purpose, as it is shown in 14 we define the member **Gravity** that, with the aid of the function "IIF" which assigns zero value to Crime Gravity "null", and its own value to the remaining instances. This makes it easier for us to continue the work below, managing the cases in which the numerator is null (inducing errors). **TotalCrimeGravity** goes to calculate Crime Gravity for each type of firearm. This newly explained member and the previous one, are useful for calculating the percentage in the member **Percentage** as you can see in 14 at the line of code 3.



	Gravity	Percentage
10mm	132	0.06%
12 gauge	3426	1.65%
16 gauge	114	0.05%
20 gauge	661	0.32%
22 LR	10851	5.22%
223 Rem [AR-15]	4806	2.31%
25 Auto	1995	0.96%
28 gauge	9	0.00%
300 Win	62	0.03%
30-06 Spr	159	0.08%
30-30 Win	288	0.14%
308 Win	294	0.14%
32 Auto	1420	0.68%
357 Mag	2633	1.27%
38 Spl	5365	2.58%
380 Auto	5974	2.88%
40 SW	8140	3.92%
410 gauge	343	0.17%
44 Mag	614	0.30%
45 Auto	6934	3.34%

Figure 15: Second query MDX, output

The output, therefore, shows the gravity in integers and the percentage of gravity. The figure 15 does not show the totality of the obtained report, but what turns out interesting is that the weapon "Handgun" is extremely used (79000 cases, approximately) for this it is attested to 38% of gravity percentage. It is, therefore, clear that this result is obtained given the great "popularity" and ease of retrieval of this weapon.

### 4.0.3 Assignment 3

*Show the incidents having a total gravity score greater or equal to the average gravity score in each state.*

```
with member avg_per_incident as
avg([Incident].[Incident Id].[Incident Id],[Measures].[Crime Gravity])
, format_string = '#,##0.00'

select [Measures].[Crime Gravity] on columns,
NON EMPTY filter([Incident].[Incident Id].[Incident Id],[Geography].[State].[State]), [Measures].[Crime Gravity] >= avg_per_incident) on rows
from [Police_Group_137]
```

Figure 16: Query MDX, third

This query was, the most complicated one in terms of computational complexity as this query required almost 52 minutes to run. It was tested various queries and methods to obtain the result and the most majority resulted in 'Out of memory' and the query was not completed. This could be because of the huge amount of **incidents** and their combination with **states**. In the next figure ?? we are presenting a piece of the current output from the functional query. The complete output consist of 38837 rows. On the left column it can be seen the incident\_id, in the middle the state where it's located and their current crime\_gravity value which is greater than the average.

		Crime Gravity
2	Ohio	4
4	North Carolina	4
6	New Mexico	6
12	Louisiana	2
15	Tennessee	2
16	California	6
18	Louisiana	6
20	Delaware	4
22	California	4
28	Georgia	4
30	California	12
32	Mississippi	2
34	District of Columbia	12
36	New York	4
42	Missouri	4
53	Ohio	2
54	Washington	4
56	Louisiana	10
61	Pennsylvania	3
62	Tennessee	4
63	Maryland	2

Figure 17: Query MDX, third

#### 4.0.4 Assignment 4

The fourth assignment required performing a geographical representation of the crime condition in the various American states, by age cluster. As you can see from figure 18, crime in adulthood occupies the most important part of the 50 pie charts in the image. From a slight observation, it appears that the states to the "East Coast" are those most affected by a high level of crime but especially widespread. Observing also the situation in California and Texas (states which are distant from the Atlantic coast), we find a high level of crime. We believe that the problem of crime should be promoted mainly in the context of economic well-being in which it is easier to buy weapons. It is also noted how the crime in the group "Teen 12-17" is slightly greater on the Atlantic coast, note such as the "clove" of this group of criminals, is lower in states bordering the Pacific Ocean and inland states than in states bordering the Atlantic.

##### Crime Gravity per State e Participant Age Group

Participant Age Group ● Adult 18+ ● Child 0-11 ● Teen 12-17

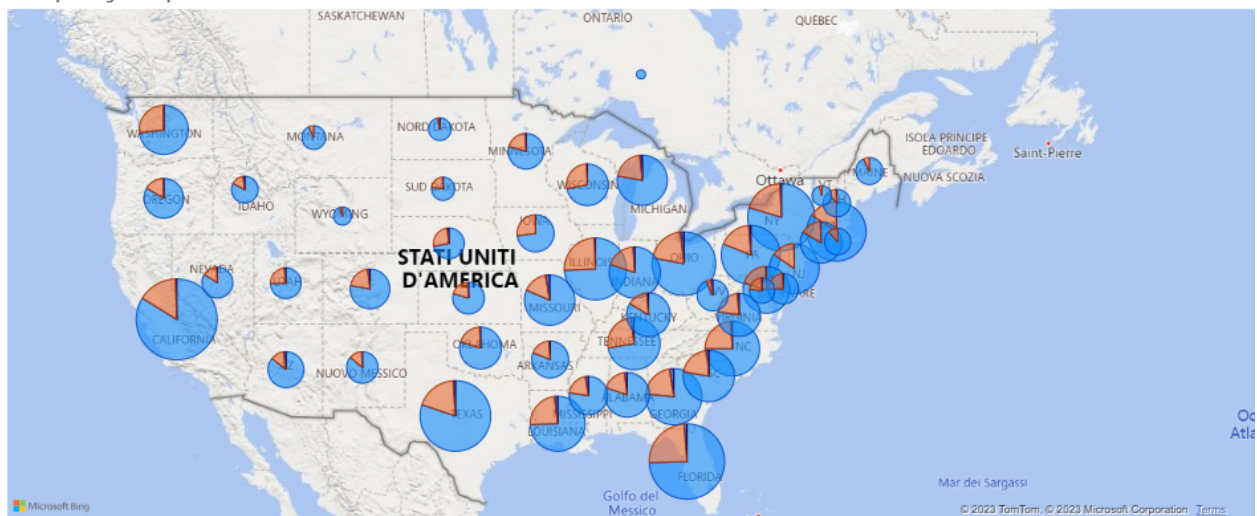


Figure 18: Geographical Distribution, Crime gravity per age group

The visualization in figure 18 is not total, the situations of Alaska and Hawaii that we have omitted for a reason of interpretability of the plot in figure above are not displayed. Both states are characterized by very low number of cases.

4.0.5 Assignment 5

We, therefore, decide to continue the analysis started and presented in figure 18 investigating which are the states with the highest level of crime, for the Participant Age group. As can be seen in the figure 19 the states with the highest crime rates are California, Florida and Texas. Florida presents a greater cardinality than cases performed by teenagers.

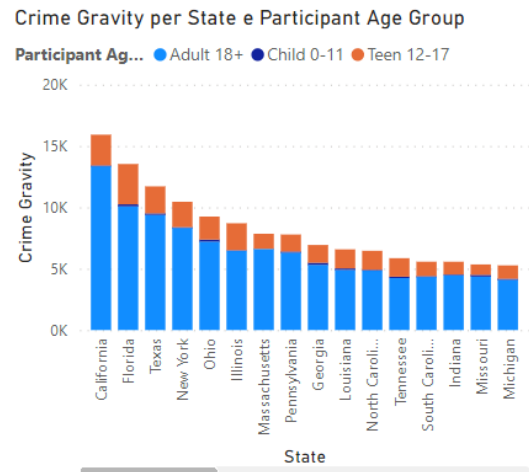


Figure 19: Crime gravity per State and Participant Age group

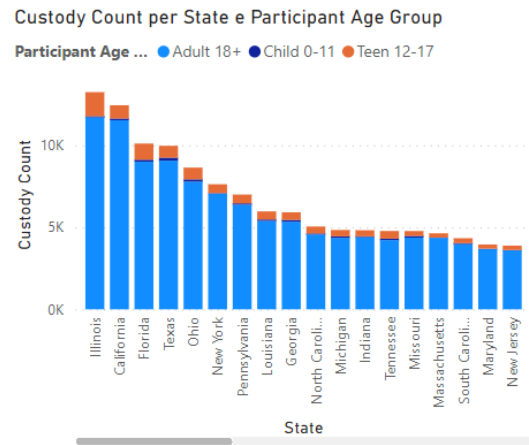


Figure 20: Custody count per State and Participant Age group

To continue the analysis in this sense, we therefore investigated how many were the cases (rather than the crime gravity) by state, and the state with the highest number of cases is Illinois. So, more crimes are committed, but they tend to be less serious in this state. Intrigued, we therefore decided to perform the same analysis but in function of the distinct cities and in accordance with the plot in figure 20 Chicago (capital of the state of Illinois) has many crimes committed. As can be seen in the figure 22, Chicago stands at a level of 9,000 cases followed by the city of New York where about 3,000 are registered.

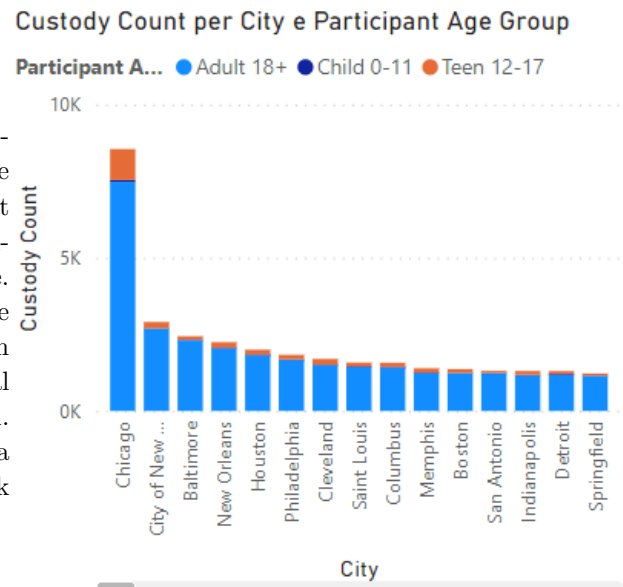


Figure 21: Histogram



Noting that the most influential cities are Chicago, New York and Baltimore, we make a comparison between the number of cases for each of these three cities and crime gravity. As can be seen from the figure 22, Chicago turns out to have a high number of cases with almost conforming gravity, unlike New York where instead the number is decidedly lower in all the years analyzed, but crime gravity is much higher: New York is therefore a city characterized perhaps by more "heinous" crimes, by a greater crime. The same applies to Baltimore, with particular reference to the year of 2017. All three states share a growing trend from 2013 to 2017, except for a substantial decrease from 2018, evidently motivated by a lack of data collected.

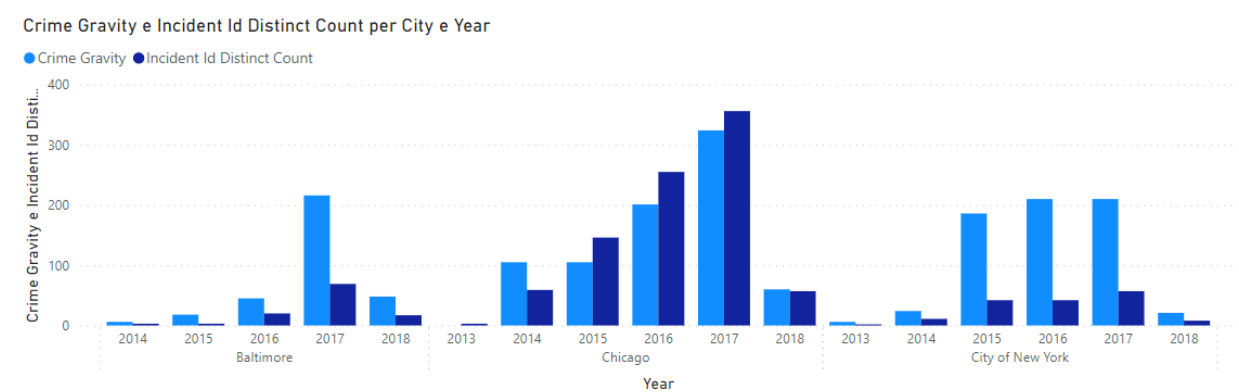


Figure 22: Crime Gravity and Incident Id Distinct Count per City and Year, comparison between three cities

We also investigate the gravity of custody by month and state, we focus only in the first 4 states such as California, Florida, Illinois and the state of New York. As you can see from the figure 23 the time trend does not vary with the change of months. A low inflection is noted for California in April, we can not give meaning to this decrease.

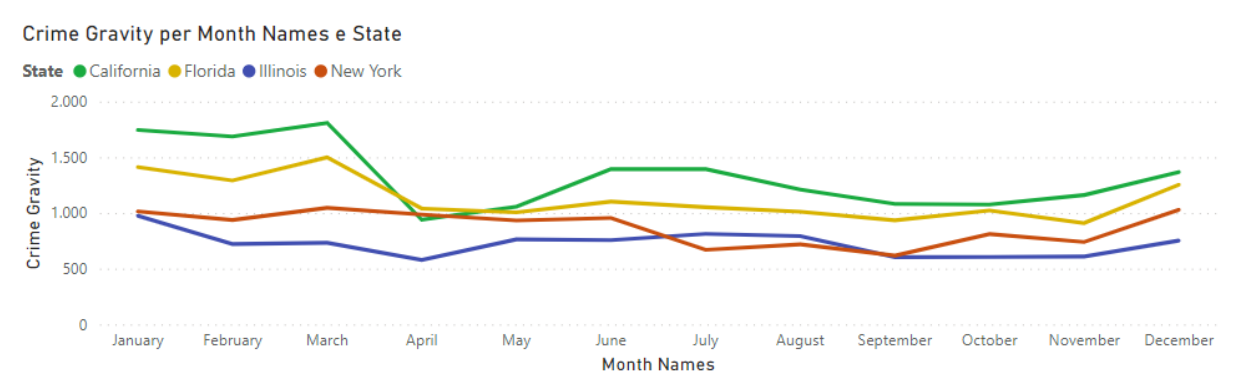


Figure 23: Crime Gravity per months and State, comparison between three States